# Homework #3 Submission

## R Script (Code)

#

# Course: IST687

# Name: Joyce Woznica

# Homework 3 - Cleaning/Munging Dataframes

# Due Date: 1/27/2019

# Date Submitted:

#

# read in a dataset so that it can be useful.

#

# Step 1: Create a function (named readStates) to read in a CSV file into R

# 1) read from a URL:

# http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-01.csv

# 2) The file is a dataset on state populations (within the United States)

readStates <- function(URL4csv)

{

# now store that in the desired readStates dataframe

# try something like

# dfstates<-readstates(urlname)

newFrame <- read.csv(url(URL4csv))

# Step 2: Clean the dataframe

# 3) Note the issues that need to be fixed (removing columns, removing rows,   
 # changing column names)

# ANSWER: Issues found: blank columns, poor column names, blank rows, factors,

# not numbers, etc.

#str(newFrame)

# remove the first 8 rows using the following

newFrame<-newFrame[-1:-8,]

# now remove the last 5 columns (5 variables)

# check what is in these columnts

#summary(newFrame[,6:10])

# reduce readStates to just having the first 5 columns

newFrame<-newFrame[,1:5]

# review what we have now

#tail(newFrame,5)

# this is unneeded information as well

newFrame<-newFrame[-52:-58,]

# 4) Within your function,

# make sure there are 51 rows (one per state + the District of Columbia)

# ANSWER: Verified with newFrame[,1]

# make sure there are only 5 columns with the columns having the following names:

# stateName, base2010, base2011, Jul2010, Jul2011

# ANSWER: set newFrame to only first 5 columns with newFrame[,1:5]

# rename the columns

# first store the existing column names in a variable for manipulation

cnames<-colnames(newFrame)

# change the first location in this vector to the statename

cnames[1]<-"stateName"

# change remaining columns

cnames[2]<-"base2010"

cnames[3]<-"base2011"

cnames[4]<-"Jul2010"

cnames[5]<-"Jul2011"

# update readStates with the new column names

colnames(newFrame)<-cnames

# now fix the stateName to remove the leading blanks and the '.'

newFrame$stateName<- gsub("\\.","",newFrame$stateName)

# 5) Make sure the last four columns are numbers (not strings)

# now replace the ',' on each number

newFrame$base2010<-gsub(",","", newFrame$base2010)

newFrame$base2011<-gsub(",","", newFrame$base2011)

newFrame$Jul2010<-gsub(",","", newFrame$Jul2010)

newFrame$Jul2011<-gsub(",","", newFrame$Jul2011)

# now update as numeric, not string

newFrame$base2010<-as.numeric(gsub(" ","",newFrame$base2010))

newFrame$base2011<-as.numeric(gsub(" ","",newFrame$base2011))

newFrame$Jul2010<-as.numeric(gsub(" ","",newFrame$Jul2010))

newFrame$Jul2011<-as.numeric(gsub(" ","",newFrame$Jul2011))

# reset the row numbers

rownames(newFrame)<-NULL

return(newFrame)

}

# now use the function to create a cleaned dataframe called dfStates

# first I will set up the URL to read

URL2Read <- "http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-01.csv"

# Step 3: Store and Explore the dataset

# 6) Store the dataset into a dataframe called dfStates

dfStates<-readStates(URL2Read)

# 7) Test your dataframe by calculating the mean for the July2011 data, by doing

# mean(dfState$Jul2011)

# you should get an answer of 6,109,645  
# ANSWER: 6,109,645

mean(dfStates$Jul2011)

# Step 4: Find the state with the highest population

# 8) Based on the July2011 data, what is the population of the state with the highest population?

# What is the name of that state?

# ANSWER: 37,691,912 - California

maxJulPop<-max(dfStates$Jul2011)

# two ways to do this, but would like to change the row names to be the state names –

# would be better

MaxRowNum<-which.max(dfStates$Jul2011)

dfStates$stateName[MaxRowNum]

dfStates$Jul2011[MaxRowNum]

# also provides the maximum population (but not the state)

dfStates[dfStates$Jul2011 == maxJulPop, "Jul2011", drop = FALSE]

# 9) Sort the data in increasing order based on the July2011 data

dfStatesSorted <- dfStates[order(dfStates$Jul2011),]

#Step 5: Explore the distribution of the states

# 10) Write a function that takes two parameters. The first is a vector and the second is a number

# 11) The function will return the percentage of the elements within the vector that is less than

# the same (i.e. the cumulative distribution below the value provided)

# 12) For example, if the vector had 5 elements (1,2,3,4,5) with 2 being the number passed

# into the function, the function would return 0.2 (since 20% of the numbers were below 2)

percentLess <- function(theVector,theNumber)

{

# for cumulative percentage, need length of the vector

vecLength<-length(theVector)

# find all the columns that are less and divide that by the total length

thePercent<-length(theVector[theVector < theNumber, drop=FALSE])/vecLength

return(thePercent)

}

# 13) Test the function with the vector 'dfStates$July2011Num' and the mean of

# dfStates$July2011Num

# answer is about 67%

percentLess(dfStates$Jul2011,mean(dfStates$Jul2011))

# to look more like a true percentage, can format this

sprintf("%1.2f%%", 100\*percentLess(dfStates$Jul2011,mean(dfStates$Jul2011)))

# There are many ways to write the function, show some examples and select the best method

# could also do an index and then store that and count the number of TRUES returned

# did this with a while loop - lots of overhead and storage of variables, but works

percentLess2 <- function(theVector,theNumber)

{

vecLength <-length(theVector)

totalMatch <- 0 # for the matches

index<-1 # for the index of the vector

while (index <= vecLength)

{

if (theVector[index]<theNumber) totalMatch<-totalMatch+1

index<-index+1

}

thePercent<-totalMatch/vecLength

return(thePercent)

}

# short and to the point, limited variable storage

percentLessBest <- function(theVector,theNumber)

{

thePercent<-length(which(theVector<theNumber))/length(theVector)

return(thePercent)

}

# I would conclude that my function "percentLessBest" is the best function for this purpose

# no additional variable storage overhead and no loops

sprintf("%1.2f%%", 100\*percentLessBest(dfStates$Jul2011,mean(dfStates$Jul2011)))

## Console Log (Executed Code)

|  |
| --- |
| > #  > # Course: IST687  > # Name: Joyce Woznica  > # Homework 3 - Cleaning/Munging Dataframes  > # Due Date: 1/27/2019  > # Date Submitted:  > #  > # read in a dataset so that it can be useful.  > #  > # Step 1: Create a function (named readStates) to read in a CSV file into R  > # 1) read from a URL:  > # http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-01.csv  > # 2) The file is a dataset on state populations (within the United States)  >  > readStates <- function(URL4csv)  + {  + # now store that in the desired readStates dataframe  + # try something like  + # dfstates<-readstates(urlname)  + newFrame <- read.csv(url(URL4csv))  +  + # Step 2: Clean the dataframe  + # 3) Note the issues that need to be fixed (removing columns, removing rows, changing column names)  + # ANSWER: Issues found: blank columns, poor column names, blank rows, factors, not numbers, etc.  + #str(newFrame)  + # remove the first 8 rows using the following  + newFrame<-newFrame[-1:-8,]  + # now remove the last 5 columns (5 variables)  + # check what is in these columnts  + #summary(newFrame[,6:10])  + # reduce readStates to just having the first 5 columns  + newFrame<-newFrame[,1:5]  + # review what we have now  + #tail(newFrame,5)  + # this is unneeded information as well  + newFrame<-newFrame[-52:-58,]  +  + # 4) Within your function,  + # make sure there are 51 rows (one per state + the District of Columbia)  + # ANSWER: Verified with readStates[,1]  + # make sure there are only 5 columns with the columns having the following names:  + # stateName, base2010, base2011, Jul2010, Jul2011  + # ANSWER: set readStates to only first 5 columns with readStates[,1:5]  + # rename the columns  + # first store the existing column names in a variable for manipulation  + cnames<-colnames(newFrame)  + # change the first location in this vector to the statename  + cnames[1]<-"stateName"  + # change remaining columns  + cnames[2]<-"base2010"  + cnames[3]<-"base2011"  + cnames[4]<-"Jul2010"  + cnames[5]<-"Jul2011"  + # update readStates with the new column names  + colnames(newFrame)<-cnames  + # now fix the stateName to remove the leading blanks and the '.'  + newFrame$stateName<- gsub("\\.","",newFrame$stateName)  +  + # 5) Make sure the last four columns are numbers (not strings)  + # now replace the ',' on each number  + newFrame$base2010<-gsub(",","", newFrame$base2010)  + newFrame$base2011<-gsub(",","", newFrame$base2011)  + newFrame$Jul2010<-gsub(",","", newFrame$Jul2010)  + newFrame$Jul2011<-gsub(",","", newFrame$Jul2011)  + # now update as numeric, not string  + newFrame$base2010<-as.numeric(gsub(" ","",newFrame$base2010))  + newFrame$base2011<-as.numeric(gsub(" ","",newFrame$base2011))  + newFrame$Jul2010<-as.numeric(gsub(" ","",newFrame$Jul2010))  + newFrame$Jul2011<-as.numeric(gsub(" ","",newFrame$Jul2011))  + # reset the row numbers  + rownames(newFrame)<-NULL  + return(newFrame)  + }  > # first I will set up the URL to read  > URL2Read <- "http://www2.census.gov/programs-surveys/popest/tables/2010-2011/state/totals/nst-est2011-01.csv"  >  > # Step 3: Store and Explore the dataset  > # 6) Store the dataset into a dataframe called dfStates  > dfStates<-readStates(URL2Read)  >  > # 7) Test your dataframe by calculating the mean for the July2011 data, by doing mean(dfState$Jul2011)  > # you should get an answer of 6,109,645  > mean(dfStates$Jul2011)  [1] 6109645  >  > # Step 4: Find the state with the highest population  > # 8) Based on the July2011 data, what is the population of the state with the highest population?  > # What is the name of that state?  > maxJulPop<-max(dfStates$Jul2011)  > # two ways to do this, but would like to change the row names to be the state names - would be better  > MaxRowNum<-which.max(dfStates$Jul2011)  > dfStates$stateName[MaxRowNum]  [1] "California"  > dfStates$Jul2011[MaxRowNum]  [1] 37691912  > # also provides the maximum population (but not the state)  > dfStates[dfStates$Jul2011 == maxJulPop, "Jul2011", drop = FALSE]  Jul2011  5 37691912  >  > # 9) Sort the data in increasing order based on the July2011 data  > dfStatesSorted <- dfStates[order(dfStates$Jul2011),]  >  > #Step 5: Explore the distribution of the states  > # 10) Write a function that takes two parameters. The first is a vector and the second is a number  > # 11) The function will return the percentage of the elements within the vector that is less than  > # the same (i.e. the cumulative distribution below the value provided)  > # 12) For example, if the vector had 5 elements (1,2,3,4,5) with 2 being the number passed  > # into the function, the function would return 0.2 (since 20% of the numbers were below 2)  >  > percentLess <- function(theVector,theNumber)  + {  + # for cumulative percentage, need length of the vector  + vecLength<-length(theVector)  + # find all the columns that are less and divide that by the total length  + thePercent<-length(theVector[theVector < theNumber, drop=FALSE])/vecLength  + return(thePercent)  + }  >  > # 13) Test the function with the vector 'dfStates$July2011Num' and the mean of dfStates$July2011Num  > # answer is about 67%  > percentLess(dfStates$Jul2011,mean(dfStates$Jul2011))  [1] 0.6666667  > # to look more like a true percentage, can format this  > sprintf("%1.2f%%", 100\*percentLess(dfStates$Jul2011,mean(dfStates$Jul2011)))  [1] "66.67%"  >  > # There are many ways to write the function, show some examples and select the best method  > # could also do an index and then store that and count the number of TRUES returned  > # did this with a while loop - lots of overhead and storage of variables, but works  > percentLess2 <- function(theVector,theNumber)  + {  + vecLength <-length(theVector)  + totalMatch <- 0 # for the matches  + index<-1 # for the index of the vector  + while (index <= vecLength)  + {  + if (theVector[index]<theNumber) totalMatch<-totalMatch+1  + index<-index+1  + }  + thePercent<-totalMatch/vecLength  + return(thePercent)  + }  >  > # short and to the point, limited variable storage  > percentLessBest <- function(theVector,theNumber)  + {  + thePercent<-length(which(theVector<theNumber))/length(theVector)  + return(thePercent)  + }  >  > # I would conclude that my function "percentLessBest" is the best function for this purpose  > # no additional variable storage overhead and no loops  > sprintf("%1.2f%%", 100\*percentLessBest(dfStates$Jul2011,mean(dfStates$Jul2011)))  [1] "66.67%" |
|  |
| |  | | --- | | > | |